**Portfolio 1 – 319**

**Connection Group Finder**

This web app allows users to search and sign up for connection groups (small groups which meet as a part of the Salt company) on and around campus.

Josh Wassenaar and Connor Ruggles

**Table of Contents**

Title Page 1

Table of Contents 2

Overview 3-4

New and Complex 5-6

Bloom’s Taxonomy 7-9

**Overview – Connection Group Finder**

The purpose of this web app is to allow for students to search and sign-up for connection groups in their area. Connection groups are small groups that meet once weekly as a part of the Salt Company – a college ministry which is a part of Cornerstone Church. This website contains a Google map API and a side panel. The map is centered right on Iowa State’s campus and contains markers that signify the location of one or more connection groups. The side panel is populated with “C-Group Cards”. These cards contain information about that specific c-group such as meeting time, location, group leaders, and contact information as well as a “Join” button.

A user can navigate the Google map by clicking and dragging, clicking one of ten buttons just below the map, or by entering in an address into the “entered address field.” Each button re-centers the map to the stated location, usually common living areas such as Richardson Court, or West Ames. One button allows the map to center on the users current location (in the event they have given access to the browser to track their current location). When a button is clicked the map’s center is changed as well as the C-Group Cards that are currently populating the side panel, the C-Group Cards will be ordered by the relative distance to the center of the map. In the event a marker on the Google map is clicked the C-Group Cards will re-order in a similar fashion.

The side panel displays multiple C-Group Cards at a time – roughly 5 – and is vertically scrollable. This allows for large number of Cards to be easily accessible without clogging up the site visually. Each card has a number of attributes on it (as was previously stated) but most notably it has a join button. When clicked a modal will be brought up with 6 fields – name, email, phone number, living area, gender, and year. Each field is required and if the form is submitted with an erroneous field the submission will be rejected and the field in question will be highlighted red. The year field is a dropdown with several options, one being “other” if other is selected another field asking the user to explain their situation will appear and must be filled out for the form to submit. In the event of a successful submission that user will be added to whichever connection group they signed up for and their information will be stored in a database.

The application needs to be run in a pretty specific way. First, extract the full CS319 directory to some place on your file system. Then, run this command in a terminal from the CS319 directory: “python -m SimpleHTTPServer 8000”. Then, go into a web browser and navigate to: “localhost:8000/projects/Portfolio1/page.html”. That will get the project running.

It will ask for authentication, and if you use your ISU email address you will be able to access it from this link: https://docs.google.com/spreadsheets/d/1UlBlLoto8QNT3558MwLjgbcvuhQUH9GXQmspBaoVaJ8/edit?usp=sharing

The reason this is so oddly specific is that the Sheets API requires not only for the developer to have access to the sheet and have API keys and things like that, but also for whoever is running the application to have access to the sheet. So unless the instructions are followed closely, you won't be able to run our application.

**New and Complex**

Some new things we worked on were working with pop-up forms, which highlight the fields that are required with a color depending on whether or not the input is valid, which is an extension of lab 2, creating and working with global custom objects, and working a lot with Google API's. Working with the API's was by far the most complex thing we did.

We had some difficulty working with the Google Sheets API, really just because of portability issues but we got those figured by re-creating our HTML file and adding in the authentication steps that were required one at a time until we got all of them working how they were supposed to. After getting the authentication to work we found that moving information around was fairly simple, it was just creating the custom objects from the data in the spreadsheet and then working with those objects however we wanted.

Specifically, we grabbed the entire sheet from Google and iterate through its rows, which is returned as an array. Those rows' columns are represented by the indexes of that array, and each row represented all of the information needed for one object, and so you grab one row at a time and get each needed cell of information and add that to it's corresponding property of the object.

Then after you have grabbed an entire rows' worth of information and put it into an object, we put that object into a global array that stores all of the objects from that sheet, and then when we want to work with the objects we either iterate through the array or sort through it depending on the property we are sorting by. This allowed us to manipulate the objects easily and made for a simple creation of “C-Group Cards”. These C-Group Cards then populated the side panel and really are the result we were looking for when we got connected to the Google spreadsheet.

Other complexity can be found in our form validation process. The form is opened up in a modal when a join button is clicked, and then the user is presented 6 fields to fill out. Each field is required and some have certain patterns that must be followed as was done in lab, in the event any of them is not filled out correctly the form will not submit and the invalid fields will be highlighted in red. When corrected the red will immediately change to blue. In addition if the option “other” is selected in the year dropdown a 7th field appears which prompts the user to explain their specific situation.

Another point of complexity exists within the Google maps API. Within the Google maps API we created markers to signify the locations of connection groups on and off of campus. When the markers are clicked they will cause the side panel to populate with different connection groups in that respective area. This is a very nice piece of functionality that makes the website easy to navigate and required the combination of Google maps API as well as our created objects from the spreadsheet that we connected our database to.

**Bloom’s Taxonomy**

When looking at different aspects of our project it is clear that we did indeed meet the requirements set for us to achieve critical thinking which include analyzing, evaluating, and creating.

**Analyzing:**

We made important distinctions between different sections of this website, and between components which worked together to accomplish some pretty cool things. First we made a clear distinction between style and functionality. Much of our style was formatted using html, css and the bootstrap framework. This allowed for us to separate the JavaScript, which we used to implement all of our functions, database connections, and API accommodations.

As for being able to distinguish between different styles of work that we needed to complete I see a number of different styles of work that we did, being Google maps, Google sheets, object manipulation, and form validation. These each had their own difficulties to overcome, for Google maps learning the API was difficult and forced us to learn how to accommodate to a different library of functions that were foreign to us. In addition, this is quite different than everything else I have worked with being that this is a totally graphical response to any functional input.

One notable distinction about the Google sheets API is the authentication requirements, which needed to be in place for us to access the information. This was by far the most annoying part of our development and forced us to shed many tears (just kidding we were fine). Through this frustrating process we now have a much greater appreciation for all things authentication and I now see how powerful security systems really are.

Object manipulation was one of the biggest things, what we did was take the information we pulled from the Google sheets we connected to and we pushed each row into a created object. We then manipulated this object to display the necessary information in a easily readable manner. This is clearly different from both APIs that we worked with and really is one of the most important concepts that allows our webpage to function in any useful manner.

As for the form validation, we were able to work with gathering and [eventually] getting to the point where we can use that user data. The form worked well and the information can all be easily stored in an array object in the JavaScript in the webpage, however we did not get the students information to be sent to the spreadsheet, as we ran out of time and could not get authentication to work until very recently.